Collaborative Learning for Recommender Systems

Wee Sun Lee LEEWS@COMP.NUS.EDU.SG
Department of Computer Science, National University of Singapore, Singapore 117543

Abstract

Recommender systems use ratings from users on items such as movies and music for the purpose of predicting the user preferences on items that have not been rated. Predictions are normally done by using the ratings of other users of the system, by learning the user preference as a function of the features of the items or by a combination of both these methods.

In this paper, we pose the problem as one of collaboratively learning of preference functions by multiple users of the recommender system. We study several mixture models for this task. We show, via theoretical analyses and experiments on a movie rating database, how the models can be designed to overcome common problems in recommender systems including the new user problem, the recurring startup problem, the sparse rating problem and the scaling problem.

1. Introduction

The growth of the internet has resulted in an increasing need for personalized information filtering systems. Recommender systems (Resnick & Varian, 1997) are systems that are designed to predict a user's preferences using features of the items and ratings given by other users. To be effective, a recommender system must deal well with the problems listed below.

New User Problem: To be able to make accurate predictions, the system must first learn the user's preferences from the ratings that the user makes. If the system does not show quick progress, a user may lose patience and stop using the system.

Recurring Startup Problem: New items are added regularly to recommender systems. A system that relies solely on users' preferences to make predictions would not be able to make accurate predictions on these items. This problem is particularly severe with systems that receive new items regularly, such as an online news article recommendation system.

Sparse Rating Problem: In any recommender system, the number of ratings already obtained is very small compared to the number of ratings that need to be predicted. Effective generalization from a small number of examples is thus important. This problem is particularly severe during the startup phase of the system when the number of users is small.

Scaling Problem: Recommender systems are normally implemented as a centralized web site and may be used by a very large number of users. Predictions need to be made in real time and many predictions may potentially be requested at the same time. The computational complexity of the algorithms needs to scale well with the number of users and items in the system.

Systems that use only the ratings of the users are usually called collaborative filtering systems. One common method of performing collaborative filtering is to use memory-based methods (Breese et al., 1998). Memory-based collaborative filtering algorithms are algorithms like nearest neighbour that predict the rating of an item based on the ratings of users who are similar to the active user. These methods can suffer from scaling problems as nearest neighbour methods can be computationally expensive. Memory based collaborative filtering algorithms also suffer from the recurring startup problem as predictions are made by using only ratings made by other users on the same items.

One way of reducing the effect of the recurring startup problem is to use the features or content of the item for prediction. If the prediction of an item depends purely on the content, then its rating would not be affected by the fact that no other user has seen the item. However, it may not be possible to learn the preference function of a user quickly without taking advantage of information obtainable from the ratings of similar users who are already using the system. As a result, a purely content based system may suffer badly from the new user problem. An even more serious drawback to purely content based methods is that even the state of the art methods are usually not able to capture subjective qualities that affect ratings in domains such as movies and music.

In this paper, we consider preference functions that encode both the effects of the content and the ratings of similar users. We then pose the recommender system problem as one of collaboratively learning each individual's preference function by all the users of the system. We study the use of finite mixture models for this task in an online learning framework. We describe theoretical analysis to understand the effects of using different types of mixture models. We also describe experiments using online approximations to the expectation maximization (EM) algorithm on a movie data set. Our results show that collaborative learning outperforms individual learning when learning only functions of the content, reducing the effect of the new user problem. We also find that learning preference functions that encode both the effects of the content as well as the ratings of similar users performs better than learning functions that use only the content. It also performs better than learning functions that use only the ratings of similar users. In particular, inclusion of the movie features improves performance under the conditions of recurring startup. All the algorithms scale very well as they take constant time to make a prediction as well as to update the model after a rating is received regardless of the number of users and items in the system.

1.1 Related Work

Early work on collaborative filtering was done by (Resnick et al., 1994) who used a memory based algorithm where similarity between users is measured using the correlation coefficient between the two users. An empirical comparison of several collaborative filtering algorithm was done in (Breese et al., 1998) where it was shown that the memory based correlation coefficient algorithm performs well. Collaborative filtering was posed as a classification problem in (Billsus & Pazzani, 1998) where a neural network was trained to do the classification. The sparse rating problem was handled in (Billsus & Pazzani, 1998) by using singular value decomposition. Recommendation was again posed as classification in (Basu et al., 1998) where it was shown that appropriately chosen content features can improve performance compared to pure collaborative filtering.

In an approach with the same underlying philosophy as ours, (Condliff et al., 1999) designed a Bayesian mixed effect model using both content and collaborative features. However, their experiments showed mixed results on the effectiveness of their model. In machine learning, the approach of learning shared parameters has previously been investigated in (Baxter, 1995) in the context of learning multiple related tasks. The general philosophy of these (and our) approaches is related to hierarchical Bayesian modeling in statistics (Gelman et al., 1995).

Models that cluster both the users and the items have been proposed in (Ungar & Foster, 1998; Hofmann & Puzicha, 1999) to handle the sparse rating problem. We show in this paper that these methods also handle the new user and recurring startup problems well. Collaborative filtering methods with provable performance bounds have been investigated in (Nakamura & Abe, 1995; Goldman et al., 1993). These algorithms have attractive performance bounds but may need modifications in order to be practically scalable. A different approach that concentrates on ordering the items was proposed in (Cohen et al., 1999) while (Pennock et al., 2000) studied the axiomatic foundations of collaborative filtering.

We describe the advantages of collaborative learning over individual learning and also the framework that we use for analysis in Section 2. The theoretical analyses for several mixture models are described in Section 3. Online approximation algorithms for learning the mixture models are described in Section 3.1. Experimental results are given in Section 4.

2. Preliminaries

We use the term collaborative learning to describe the situation where the ratings of other users who are similar to the active user are utilized in learning the parameters of the active user's preference function. The features used for learning the preference functions of the users can include those that describe the content of the items as well as those that encode the ratings of similar users. If the features of the content is not used, then collaborative learning is reduced to collaborative filtering. In the case where the features consist of only descriptions of the content, the ratings of similar users can help to learn the function faster, easing the new user problem. In the case where both the content features and similarity features are present, the content features can ease the recurring startup problem while the similarity features can capture some of the subjective qualities of the item not captured by the content features.

We analyse the accuracy of the systems' predictions using an online learning model. In online learning, the learning algorithm is asked to make a sequence of predictions. At time t the algorithm is allowed access to the labels of all the items up to time $t-1$ in making its prediction, and suffers some loss due to the difference between its prediction and the actual label that is encountered at time t. This framework is reasonable for studying the effectiveness of the algorithm in situations where the user is allowed to query the predicted rating for any item although it may not be the best framework for studying the effectiveness of algorithms intended for recommending items. The problem of recommending items is more complex as the item chosen for recommendation may affect what the system learns about the user. We do not consider the recommendation problem in this paper.

3. Finite Mixture Models

In this section, we describe the mixture models and some of their theoretical properties. We show how different models have different bounds on their losses when used with idealized algorithms. These bounds help describe how the different parameters of the models affect the problems described in the introduction, hence providing some guide for designing models that can overcome the problems.

We use finite discrete values for the parameters of the models in our analysis. With a fine enough discretization, the models can approximate well the corresponding models that use continuous parameters. The algorithms used for deriving the bounds are computationally impractical for these models but the analysis illustrates some of the intrinsic properties of the model classes. These properties provide some guide for helping select the models to experiment with. Our experiments in Section 4 show that the behaviour of the practical algorithms when used on continuous parameters are qualitatively similar to that described by the theoretical analysis.

We assume that each user belongs to one of M classes of users and that users in the same class all have the same preference function. Furthermore the output of each preference function is corrupted by noise. For this paper, we will consider linear function of features. The set of features will consist of indicator functions and a bias (feature that is always 1) that is used to encode the average rating of the class. An indicator function $g_A(x)$ outputs 1 if x is a member of the set A and 0 otherwise. The observed rating for class i is described by $r_i(x) = \sum_{A \in \mathcal{A}} w_{Ai} g_A(x) + b_i + \epsilon_i$ where ϵ_i is a zero mean noise and w_{Ai} are the weights of the linear function and b_i is the bias. For the movie rating experiment, our sets consist of movie categories such as thriller, drama and animation. We will also use singleton sets consisting of only one movie each to encode the group opinion of users in the class for each particular movie. One way to interprete such a function is that the bias codes the average movie ratings while each category of movie provides a correction with a further correction by the group opinion about the movie (this interpretation may not actually be correct as the functions are learned from data without such constraints).

Assume that we have a finite set of models $\mathcal{M}$ and a sequence of ratings $r^t = r_1, r_2, \ldots, r_t$. An algorithm predicts using a sequence of numbers $\hat{r_1}, \hat{r_2}, \ldots, \hat{r_t}$. We use the squared loss $L_s(r, \hat{r}) = (r - \hat{r})^2$ and the absolute loss $L_a(r, \hat{r}) = |r - \hat{r}|$ for measuring performance. Algorithms that have access only to $r_1, \ldots, r_{i-1}$ when making predictions at time i are called online algorithms. Let Opt_s, and Opt_a be the sum of losses on r^T for optimal models in the class for the squared loss and the absolute loss respectively. Algorithms based on aggregrating strategies were shown to achieve the following bounds in (Haussler et al., 1998).

Theorem 1 *Assume that $\mathcal{M}$ is a finite set of models. For a sequence of ratings $r^T = r_1, r_2, \ldots, r_T$, there exist online algorithms that achieve*

$$\sum_{i=1}^{T} L_s(r_i, \hat{r}_i) \leq Opt_s + \frac{1}{2} \log |\mathcal{M}|$$
$$\sum_{i=1}^{T} L_a(r_i, \hat{r}_i) \leq \frac{\eta Opt_l + \log |\mathcal{M}|}{\ln \frac{2}{1+e^{-\eta}}},$$

where $\eta > 0$ depends on the parameter used in the algorithm for the absolute loss.

From the theorem, we see that the performance of the models can effectively be bounded by the number of models used when the appropriate aggregrating strategies are used. For the purpose of analysis, we will assume that the weights and biases are all discretized to L levels so that we have a finite set of models. Let $W = \{w_1, \ldots, w_L\}$ be the set of possible discretized values. Define the class of models $D = \{f(x) = \sum_{i=1}^{k} w_i g_i(x) + b | w_i \in W, b \in W\}$.

Corollary 2 *Assume a system with X users and M classes of users, where each class of users predict using a model in D. For such a system, there exists online algorithms that achieve*

$$\sum_{i=1}^{T} L_s(r_i, \hat{r}_i) \leq Opt_s + \frac{1}{2}(X \log M + M(k+1) \log L)$$
$$\sum_{i=1}^{T} L_a(r_i, \hat{r}_i) \leq \frac{\eta Opt_l + X \log M + M(k+1) \log L}{\ln \frac{2}{1+e^{-\eta}}},$$

where $\eta > 0$ depends on the parameter used in the algorithm for the absolute loss.

Proof. Using Theorem 1, we only need to count the number of possible models in order to prove the bounds. There are M^X ways to assign users to classes and $|D|^M$ ways to assign functions to classes. The class D consists of L^{k+1} elements. This gives $M^X L^{(k+1)M}$ possible states of the system. Applying Theorem 1 gives the bound. □

The losses in the bound can be separated into two components. The component with $X \log M$ bounds the losses that appear because we do not know which class each user belong to. We call this the **classification loss**. The component with $M(k+1) \log L$ bounds the shared cost of estimating the parameters of the functions and we call this the **parameter estimation loss**. If each user preference function were learned independently instead of collaboratively, the loss bound by applying Theorem 1 would consist entirely of parameter estimation losses which depends on

$X(k+1)\log L$. The number of classes M is usually much smaller than the number of users X, so the total classification and estimation losses are usually smaller when collaborative learning is performed. However, the best function for the class may not perform as well as a function that is personalized for the individual user. We call this differences in performance the **approximation loss**. The approximation loss can be made smaller by increasing the number of classes M. Collaborative learning with the mixture model allows the trade-off between the estimation and classification loss on one hand and approximation loss on the other hand to be made by varying the number of classes. Furthermore, the effect of increasing the number of classes on the new user problem should be small since the classification losses grows logarithmically with M. However, it could cause serious concern on the recurring startup problem and the sparse rating problem as the bound on the parameter estimation loss grows linearly with M.

When the indicator functions for each movie are included in the feature set, the parameter estimation cost can get quite large as it depends on $M(k+1)\log L$. When the data is sparse as in the case at the startup stage of the system, the performance may be poor as a result of the large parameter estimation cost. For this case, we consider another mixture model where the items are clustered as well. We study the case where the other item features are not available (pure collaborative filtering) to show that the recurring startup problem can be mitigated even without item features. In this model, there are M user classes and N item classes resulting in MN pairs of classes. Each pair is assigned a predictor μ from a set Ω with L elements.

Corollary 3 *Assume a system with X users, Y items, M classes of users and N classes of items, where each class pair predict using a value in Ω. For such a system there exists online algorithms that achieve*

$$\sum_{i=1}^{T} L_s(r_i, \hat{r}_i) \leq Opt_s + \frac{1}{2}(X\log M + Y\log N + MN\log L)$$

$$\sum_{i=1}^{T} L_a(r_i, \hat{r}_i) \leq \frac{\eta Opt_l + X\log M + Y\log N + MN\log L}{\ln \frac{2}{1+e^{-\eta}}},$$

where $\eta > 0$ depends on the parameter used in the algorithm for the absolute loss.

Proof. As in the proof of Corollary 2, we only need to count the number of possible configurations for the system. There are M^X ways to assign users to classes, N^Y ways to assign items to classes and L^{MN} ways to assign values to class pairs. This gives $M^X N^Y L^{MN}$ possible states that the system can possibly be in. Applying Theorem 1 gives the bound. □

If MN is small, the system should suffer less from the sparse data problem. When the number of items is large, the average cost per item depends approximately on $\log N$ which is relatively small, indicating that the effect of recurring startup on the estimation losses is small.

3.1 Practical Online Algorithms

The theoretical analysis suggests several model classes that can deal well with the problems in recommender systems. Unfortunately, the algorithms used in the analysis are computationally impractical. In this section, we derive some practical algorithms for the model classes. We show experimentally in Section 4 that these algorithms retain much of the properties indicated by the theoretical analysis.

We use an online gradient descent (Widrow-Hoff) algorithm with a linear function for the case where learning is done individually and not collaboratively from the features of the item. In this case the indicator functions for each movie are redundant and are not included. The learning rate was chosen to be $1/(4\|x\|^2)$, where $\|x\|^2$ is the square of the L_2 norm of the input as recommended by (Cesa-Bianchi et al., 1996).

For the collaborative learning of the mixture model, we use an online approximation to the expectation maximization (EM) algorithm based on the Widrow Hoff algorithm. We first describe the EM algorithm to motivate our online approximation. To use the EM algorithm, we need a probabilistic model for the error of the linear functions. We assume that the errors have zero mean Gaussian distributions with unknown variance for each class. Let ψ be a d dimensional vector containing the unknown parameters of the system. In our system, these are the $M(k+2)$ unknown parameters (including the unknown variances of the errors). Let $p(R^T;\psi)$ be the probability distribution of R^T when ψ is the vector of parameters that define the distribution. Given an observation vector r^T, the likelihood function (which is a function of ψ) is $p(r^T;\psi)$. The aim of the maximum likelihood method is to find a value ψ that maximizes $L(\psi) = p(r^T;\psi)$.

With the EM algorithm the maximum likelihood problem is treated as a problem with incomplete data. We introduce the following sequence of unobservable data $\xi = \xi_1, \ldots, \xi_X$ where ξ_u is an M dimensional vector of zero-one indicator variables and where each component ξ_{iu} of ξ_u is one or zero according to whether the corresponding user u belongs to class i. Only one of ξ_{iu} equals 1 for each u. The complete data is then the sequence $w^T = (r_1, \xi_{u_t}), \ldots, (r_T, \xi_{u_T})$, where $u_t \in \{1, \ldots, X\}$ corresponds to the user at time t. Let $L_c(\psi) = p(w^T;\psi)$ be the complete data likelihood function.

The $(k+1)$th iteration of the EM algorithm is defined by the following two steps:

E-Step. Calculate $Q(\psi;\psi^{(k)}) = E\{\log L_c(\psi)|z^t;\psi^{(k)}\}$ where the expectation is taken with respect to $\psi^{(k)}$.

M-Step. Choose $\psi^{(k+1)}$ to be any value of parameters that maximize $Q(\psi;\psi^{(k)})$.

The likelihood of the parameters $\psi^{(k)}$ can be shown to be non-decreasing with respect to the iterations of the EM algorithm (Dempster et al., 1977) although it may not necessarily converge to a global minima.

For our case, the complete data likelihood function can be written as

$$\prod_{t=1}^{T}\frac{1}{M}\exp(-\sum_{i=1}^{M}\xi_{iu_t}((r_t-\sum_{A\in\mathcal{A}}w_{Ai}g_A(x_t)+b_i)^2/2\sigma_i^2+(\log 2\pi\sigma_i^2)/2)).$$

Taking log of the function followed by the expectation, we find that at iteration k we want to minimize the following function

$$\sum_{t=1}^{T}\sum_{i=1}^{M}\xi_{iu_t}^{(k)}((r_t-\sum_{A\in\mathcal{A}}w_{Ai}g_A(x_t)+b_i)^2/2\sigma_i^2+(\log 2\pi\sigma_i^2)/2)$$

where $\xi_{iu_t}^{(k)}$ is the expected value corresponding to ξ_{iu_t} at iteration k, given r^T and the current parameter values. This is just the posterior probability of user u_t belonging to class i. Assuming that each user is equally likely to belong to each of the M groups, the probability can be calculated as

$$\xi_{iu_t}^{(k)} = \frac{\prod_{u_j=u_t}p(r_j|\xi_{iu_t}=1;\psi)}{C},$$

where $C=\sum_{i'=1}^{M}\prod_{u_j=u_t}p(r_j|\xi_{i'u_t}=1;\psi)$ is a normalizing constant.

Our online approximation to the algorithm at time t consists of taking a step in minimizing

$$\sum_{i=1}^{M}\zeta_{iu_t}((r_t-\sum_{A\in\mathcal{A}}w_{Ait}g_A(x_t)+b_{it})^2/2\sigma_{it}^2+(\log 2\pi\sigma_{it}^2)/2),$$

where in an approximation to (3.1), ζ_{iu_t} is initialized to $1/M$ and updated by

$$\zeta_{iu_{t+1}} = \frac{\zeta_{iu_t}p(r_t|\xi_{iu_t}=1;\psi)}{\sum_{j=1}^{M}\zeta_{ju_t}p(r_t|\xi_{ju_t}=1;\psi)}.$$

The weights w_{Ait} (and the bias b_{it}) are updated using a gradient step $w_{Ai(t+1)}=w_{Ait}+\eta\zeta_{iu_t}l(r_t,x_t)g_A(x_t)$, where $l(r_t,x_t)=r_t-\sum_{A'\in\mathcal{A}}w_{A'it}g_{A'}(x_t)+b_{it}$ and we use $\eta=1/4||x_t||^2$ as in the Widrow Hoff algorithm.

We also attempt to approximate the maximum likelihood variance estimates by $\sigma_{i(t+1)}^2=(\sigma_{it}^2+\zeta_{iu_t}l(r_t,x_t)^2)/S_i$, where $S_i=\sum_{j=1}^{t}\zeta_{iu_j}$.

An approximate online algorithm for the pure collaborative filtering case where a different variance is used for each item is derived in a similar manner. The case where both the users and items are clustered appears to be more difficult as there does not appear to be an efficient algorithm for calculating the conditional expectation of the unobservable data in the batch case. However, we use a heuristic online approximation algorithm where we update the conditional expectations for the user class assuming that the conditional expectations for the item class are actually correct and vice versa. Details are omitted and can be found in (Lee, 2000).

4. Experiments

4.1 Data

The EachMovie[1] data set was obtained by the DEC Systems Research Center by running a movie recommendation service for 18 months. The data set contains 72916 users who entered a total of 2811983 numeric ratings for 1628 different movies. Each rating comes from the set $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ where 0.0 is the lowest score and 1.0 is the highest. The movies are categorized into 10 categories: action, animation, art foreign, classic, comedy , drama, family, horror, romance, and thriller, where a movie may belong to more than one category.

The number of users who made N or more ratings decreases faster than exponentially (see (Lee, 2000)), hence measuring the average prediction error on the whole system puts significantly more emphasis on the performance of the system on users who have not made many ratings (new users). Another relevant aspect of this data set is that the choice of movies that were rated is under the control of the user (and the recommender system that was running). Hence, some movies may have received considerably more ratings than others. There may also be other possible biases, for example, it is possible that users may rate movies that they have a strong opinion on initially and leave the average movies for later rating.

4.2 Setup

For our experiments, we perform our measurements using the absolute loss as it is intuitively easier to understand than the squared loss. Experiments for all the algorithms were done the same way. The data set of ratings was sorted according to the time the ratings were obtained. A prediction was made for item t in the sorted list before the rating was shown to the algorithm. The rating was then used to measure the absolute prediction error $|r_t-\hat{r}_t|$ where r_t is the actual rating while $\hat{r}_t$ is the predicted rating. After that, the

[1] http://research.compaq.com/SRC/eachmovie/

Predictor	Details	Avg Abs Error (all data)	Avg Abs Error (first 30000)
Average Movie Rating		0.2195	0.1676
Linear function on movie features only	trained individually	0.2683	0.2607
Linear function on movie features only	collab. training with 50 classes	0.2370± 0.00001	0.2106± 0.00006
Linear function on {movie features + movie indicator functions}	collab. training with 30 classes	0.1963± 0.00012	0.1840 ± 0.00035
User clustering (no movie features)	30 user classes	0.1980± 0.00023	0.1788 ± 0.00091
User and Item clustering (no movie features)	70 user by 70 item classes	0.1998± 0.00036	0.1649 ± 0.00027
Correlation Coeff Algorithm	7000 users as predictors	0.2050	0.1810

Table 1. Average absolute prediction error, averaged over all ratings and over the first 30000 ratings where predictions are done in an online manner. The 95% confidence intervals are given with respect to the random initializations assuming that the average values are drawn from normal distributions with unknown variances.

rating r_t was used to update the state of the algorithms before the process was repeated with item $t + 1$. We chose to sort the ratings according to the time of entry in order to simulate the way that the system actually worked (i.e. the order by which people choose to rate movies etc.). We were able to run the experiments on the entire data set of about 3 million ratings as all the algorithms run quickly.

We compared the performance of the algorithms against the memory based correlation coefficient algorithm and the prediction of the average movie rating. The average movie rating is a very simple non-personalized predictor. In order to claim that personalization is beneficial, a personalized algorithm will have to at least outperform the average movie rating used as a prediction. The correlation algorithm was introduced by Resnick et. al. in (Resnick et al., 1994) and is known to have good predictive performance. Details of our implementation are described in (Lee, 2000). In order to make the experiment feasible on the entire EachMovie data set, we limit the users that are used as predictors to the first X' users instead of using all users of the system.

The mixture model algorithms require random initializations. We initialized the biases to 0.5 and all the weights randomly between -0.1 and 0.1. For calculating the variances, σ_i^2 was initialized to 0.05 and S_i to 1. Initializations for the pure collaborative mixture models are similar and are described in (Lee, 2000). All measurements are averaged over five random initializations. In order to allow the parameters to stabilize a little, we use the average movie rating as predictors for the first 1000 ratings of the system (for all algorithms in this paper) while updating the parameters using those ratings.

We performed experiments using M=10, 30, 50 and 70 classes of users. For the case where both the users and items are clustered, we tried 10 user by 10 item groups, 30 user by 30 item groups, 50 user by 50 item groups and 70 user by 70 item groups. For the correlation coefficient algorithm, we tried using X'=1000, 3000, 5000 and 7000 users as predictors. Due to lack of space, we report on only the settings that give the best result for each algorithm on the entire data set in Table 1.

4.3 Results

From the column on the average absolute error on all data in Table 1, we see that learning a linear function of the movie features collaboratively significantly outperforms non-collaborative learning. However, neither of these algorithms outperform the average movie ratings as predictors. If we are restricted to only learning these features with a linear function, it seems that nonpersonalized prediction using the average movie ratings outperforms personalized prediction. However, pure collaborative filtering that clusters the users together outperforms the nonpersonalized average movie ratings predictor.

By adding the indicator functions for each movie, we allow the linear functions that are learned collaboratively to utilize both the features and the group preference. This improved the performance compared to pure collaborative filtering.

Clustering both users and items does not perform as well as clustering just the users. This is to be expected since the approximation error of the model that clusters both the users and the items is most probably larger then the approximation error of clustering just the users (to see this, items that are not clustered together are free to have their own average ratings).

Our collaborative filtering algorithms also outperformed the the correlation algorithm in these experiments.

From the column on the average absolute error on the first 30000 ratings in Table 1, we see that all the algorithms performed poorer than the non-personalized average movie ratings predictor except for the algorithm that clusters both the users and the movies. This is due to the sparse data problem. By having a smaller parameter estimation loss,

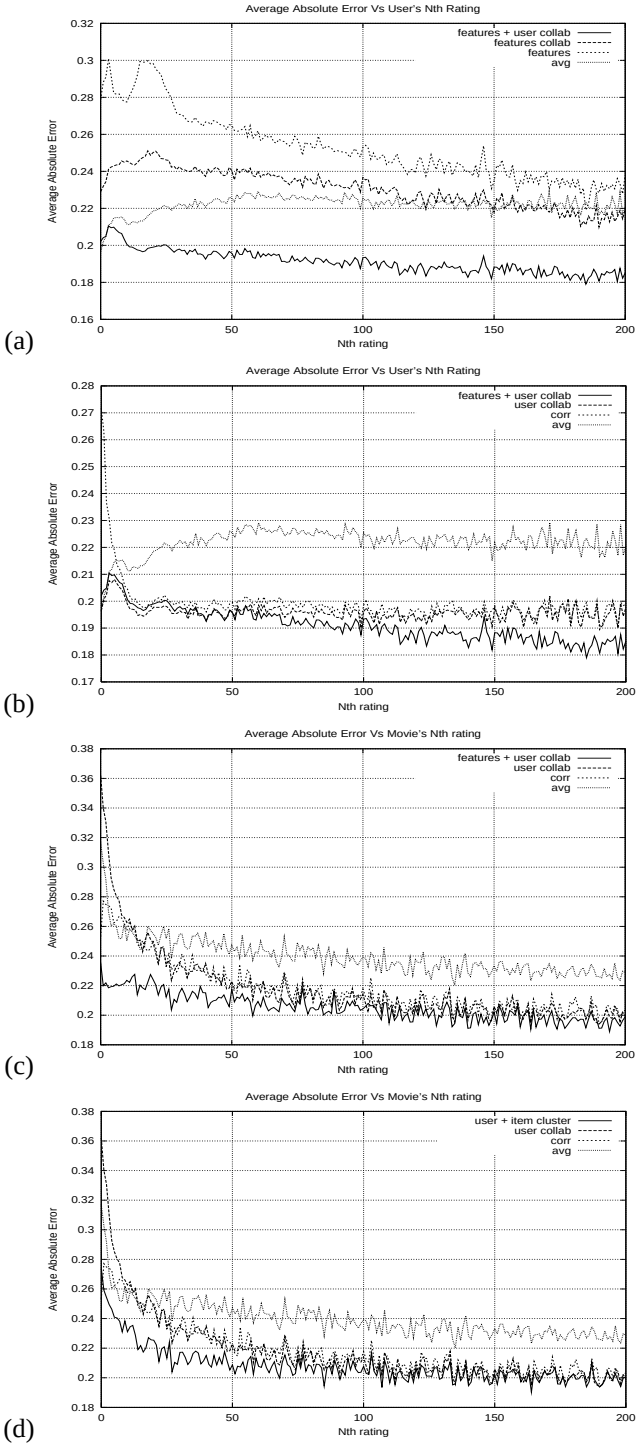

Figure 1. Top two figures show the average absolute error for the user's Nth rating. Bottom two figures show the average absolute error for the movie's Nth received rating. The label "avg" indicates using average movie rating as predictor, "features" indicates using linear predictors on features trained individually, "features collab" indicates linear predictors on features trained collaboratively, "features + user collab" indicates linear predictors on features and movie indicator functions trained collaboratively, " corr" indicates the correlation coefficients algorithm and "user + item cluster" indicates the algorithm where both the users and items are clustered.

the algorithm that clusters both the users and movies performs more effectively in these situations.

In Figure 1(a) and (b), we plot the average absolute error on the user's Nth rating averaged over all users who made N or more ratings. This gives an indication of the average prediction accuracy after a user has made N ratings. In Figure 1(a), we can see that collaborative learning of the linear function of the features significantly outperforms learning the function non-collaboratively but performs worse than the non-personalized predictor in the first 150 predictions.

Figure 1(b) compares the algorithm that collaboratively learns the movie features as well as the collaborative user opinions with the algorithm that uses only the user opinions. It shows that using the features improves on just using the user opinions. We also show the performance of the correlation coefficient algorithm. The correlation coefficient algorithm requires a certain number of ratings from a user before it gives good performance. In Table 1, we see that it suffers a relatively large loss compared to the clustering algorithm. However this is mainly because of the large number of users who makes very few ratings. From Figure 1(b), we see that in practice this may not be serious because the system performs reasonably after more than 10 ratings have been received.

In Figure 1(c) and (d), we plotted the average absolute error on the movie's Nth received rating. We see that the algorithm that combines movie features and collaborative opinion works quite well even when the movie has not received many ratings indicating it does not suffer badly under recurring startup conditions when compared to the other algorithms. The algorithm that clusters both users and movies also learns quickly under recurring startup conditions even though it does not use any information from the movie features due to its low parameter estimation loss.

4.4 Discussion

Even though the theoretical results do not directly apply to the algorithms we used, the results are qualitatively similar. Collaborative learning has much smaller new user losses compared to non-collaborative learning as predicted by theoretical considerations. Clustering both the users and movies deals better with the sparse rating problem and recurring startup problem but has poorer approximation error as predicted by the theory. We feel that theoretical considerations can help guide the search for even more effective models for recommender systems.

For good scaling behaviour, online updating was done after each new rating is received for all the mixture models. For a fixed number of clusters, the update takes constant time. Similarly, each prediction takes constant time. With 30 clusters, the best performing algorithm that utilizes both

the movie features as well as group preferences takes under 2 minutes to run on the entire data set on a Sun Ultra 60 workstation. The time complexity for each update and for each prediction using this algorithm is $O(Mk)$ where M is the number of clusters and k is the number of movie features (excluding the movie indicator functions). The algorithm that clusters both the users and items (70 user cluster by 70 item cluster) takes approximately 2 hours to run through the entire data set on the same workstation. The time complexity for each update or prediction is $O(MN)$ where M is the number of user clusters and N is the number of item clusters. Since they are performed online, the algorithms also require a very small amount of memory. However, there is no guarantee that the algorithms can find the globally optimal predictor.

Interesting directions for future work include looking for methods that perform recommendations well instead of just good predictions. Recommendations are more difficult as the items that the system recommends affect the information that the system gets about the users and the items. Hence, the system may sometimes want to recommend an item when it is not certain whether the user will like it in order to gain useful information about the user or the item.

5. Conclusions

By posing the problem of learning for recommender systems as a collaborative learning problem, improved performance can be obtained as a result of joint learning of parameters of shared functions by many users. We showed theoretically that the benefit exists for mixture models using idealized algorithms. The analysis suggests methods of designing mixture models that are able to overcome many of the common problems in recommender systems. We use the insights to design practical algorithms that work well in experiments with a movie database.

Acknowledgements

The author would like to thank the anonymous reviewers whose comments helped improve the paper. Thanks also to Mong Li Lee and Phil Long for many useful suggestions. This research was supported by by the National University of Singapore Academic Research Fund grant RP3992710. The EachMovie data set was provided by Digital Equipment Corporation.

References

Basu, C., Hirsh, H., & Cohen, W. (1998). Recommendation as classification: Using social and content-based information in recommendation. *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 714–720). Madison, WI.

Baxter, J. (1995). Learning internal representation. *Proc, of the 8th Int. Conf. on Computational Learning Theory*.

Billsus, D., & Pazzani, M. (1998). Learning collaborative information filters. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 46–54). Morgan Kaufman.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th Annu. Conference on Uncertainty in Artificial Intelligence* (pp. 43–52).

Cesa-Bianchi, N., Long, P. M., & Warmuth, M. K. (1996). Worst-case quadratic loss bounds for prediction using linear functions and gradient descent. *IEEE Tran. on Neural Networks*, *7*, 604–619.

Cohen, W. W., Schapire, R. E., & Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, *10*, 243–270.

Condliff, M. K., Lewis, D. D., Madigan, D., & Posse, C. (1999). Bayesian mixed-effects models for recommender systems. *Proceedings of Workshop on Recommender Systems: Algorithms and Evaluation*.

Dempster, A., Laird, N. M., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–38.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995). *Bayesian data analysis*. Chapman and Hall.

Goldman, S. A., Rivest, R. L., & Schapire, R. E. (1993). Learning binary relations and total orders. *SIAM J. Comput.*, *22*, 1006–1034.

Haussler, D., Kivinen, J., & Warmuth, M. K. (1998). Sequential prediction of individual sequences under general loss functions. *IEEE Trans. on Information Theory*, *44*, 1906–1925.

Hofmann, T., & Puzicha, J. (1999). Latent class models for collaborative filtering. *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (pp. 688–693).

Lee, W. S. (2000). *Online clustering for collaborative filtering* (Technical Report TRA8/00). School of Computing, National University of Singapore.

Nakamura, A., & Abe, N. (1995). On-line learning of binary and n-ary relations over multi-dimensional clusters. *Proc. 8th Annu. Conference on Comput. Learning Theory* (pp. 214–221). ACM Press, New York, NY.

Pennock, D. M., Horvitz, E., & Giles, C. L. (2000). Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. *Proceedings of the Seventeenth National Conference on Artificial Intelligence*.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*.

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, *40*, 56–58.

Ungar, L. H., & Foster, D. P. (1998). Clustering methods for collaborative filtering. *Workshop on Recommendation Systems at the Fifteenth National Conference on Artificial Intelligence*.